Computer Vision: Project 4 Write-up

Rose Ridder, Lan Ngo

Starting Out

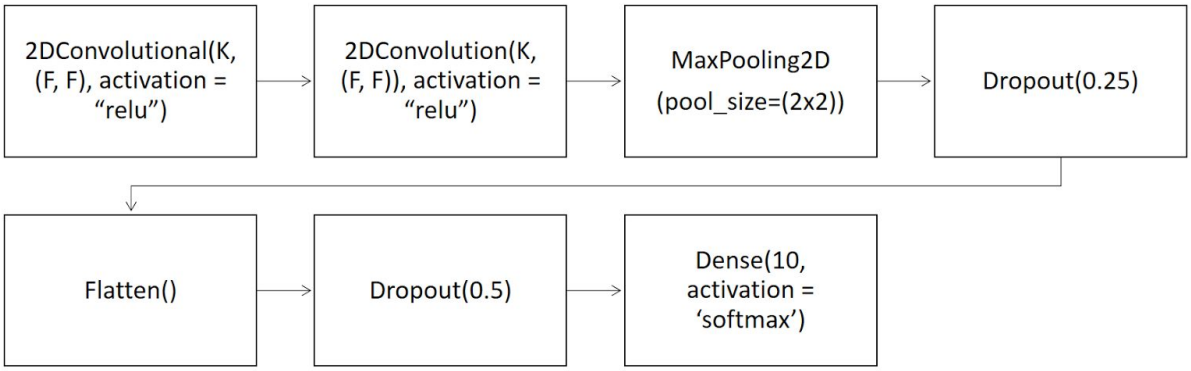
We started this lab by familiarizing ourselves with the types of neural networks that were applicable to the MNIST dataset and researching tools available to implement them. Stack Overflow was very useful for this because many comments gave alternatives, or strategies to listed implementations. When we found a package that looked applicable, we looked at the specific documentation for that sort of network.

For using Convolutional Neural Networks (CNN), we started with online tutorials and documentation using keras https://elitedatascience.com/keras-tutorial-deep-learning-in-python and learned more about what exactly the hyperparamters do by reading the keras documentation and some other sources such as Stanford's cs231 course notes. We also improve our testing process by saving the weights to a .h5 file to reduce training a new model from the beginning every time.

For the Multilayer Perceptron Classifier, we started by reading about the parameters (http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html) and then investigated tutorials and implementations on the scikit-learn site, deciding on an implementation for the MNIST data specifically at http://scikit-learn.org/stable/auto_examples/neural_networks/plot_mnist_filters.html.

Keras Convolutional Neural Net

We implemented the CNN with the architecture as illustrated by the flowchart and varied the hyperparameters such as number of receptive fields (K) and the size of the kernel (F) in in the convolutional layer, the output size of the fully-connected (Dense) layer (D) and the batch-size (B) the number input nodes evaluated at a time. Table. 1 shows the fit time, reported accuracy (accuracy on the train set in the last epoch), training and testing accuracy and evaluation time. The best performance is highlighted in bold. Our model's best performance reached 98.37% on the testing set with 5249.60s fitting time when K = 32, F = 3, D = 128 and B = 32.



Parameters	Fit Time	Reported Accuracy	Training Set Accuracy	Training Set Eval Time	Test Set Accuracy	Test Set Eval Time
K=3, F=3, D=128, B=32	226.50	94.57	98.11	7.82	97.73	1.06
K=3, F=5, D=128, B=32	235.52	93.77	95.29	8.23	95.76	1.28
K=3, F=11, D=128, B=32	291.84	85.18	92.00	12.26	92.24	1.67
K=3, F=3, D=64, B=32	195.76	92.97	95.90	8.70	95.99	.96
K=3, F=3, D=256, B=32	218.33	95.00	97.83	6.67	97.33	1.03
K=3, F=3, D=384, B=32	335.69	95.19	97.49	7.30	97.20	1.12
K=3, F=3, D=128, B=64	186.83	93.90	97.55	7.25	97.35	0.91
K=3, F=3, D=128, B=16	256.71	92.77	97.44	7.55	96.97	0.96
K =16, F=3, D=128, B=32	1131.6080	96.34	98.53	28.52	98.08	4.50
K=32m F=3, D=128,B=32	**5249.60**	**96.68**	**98.87**	**57.50**	**98.37**	**8.06**
K=32m F=3,	6807.	93.76	98.34	225.46	98.20	37.38

D=128,B=32, Batch Normalization	63					

For MLP model, we varied the architecture of the classifier by changing the activation algorithm, the number of output nodes for each layer and the number of layers. The table below summarizes our findings for various hyperparameters, with a max_iter set to 500 in order to allow all of the classifiers to reach a point where the loss did not change between iterations. Three tables are included at the end of our report documenting our findings when adjusting the max_iter value to 10,50,and 100 respectively, which we used in our original testing phases in order to get data more quickly and see which hyperparameters get higher accuracy with fewer adjustments of the weights.

We found the most successful parameters were Adam as the optimizer with the tanh function as the activation function. We further identified that the classifier with three hidden layers with 50 nodes each outperformed all the others in terms of accuracy for the test set, and was within one 2 tenths of the accuracy of the two-layer 50-node classifier for the training set.

To reach this finding, we tried several different numbers of nodes for two hidden layers, including two layers with 50 nodes, a 50->20 and a 20->50 setup, and found that the 50->20 was comparable to the 50->50 in terms of the training time and test set accuracy, but was slightly lower for the training set accuracy (its accuracy was .3 lower that the 50->50 architecture). The test set accuracies were within one 100th, at 96.75% accuracy for the 50->50 and 96.76% accuracy for the 50->20. The 20->50 structure was not as accurate.

We also tried a classifier with three hidden 50-node layers, three hidden layers of 50->20->50, and 20->50->20, and one classifier with five 50-node hidden layers. The classifier with three 50-node layers performed best with an accuracy of 97.79% for the test data. The others were around 96.1-96.7% accuracy, except for the 20->50->20 structure, which had a 95.2% accuracy.

It is important to note that for the same structures, the runtime often fluctuated by about 10 seconds during different runs. This was likely due to our computer's CPU usage by other programs at the same time, but it is important to keep in mind when identifying the best parameters and comparing the run times.

Scikit-Learn MLP:

(Max_iter = 500 to reach equilibrium for all weights)

Parameters	Fit Time	Training Set Accuracy	Training Set Eval Time	Test Set Accuracy	Test Set Eval Time
Single layer adam	28.81	93.10	0.35	92.50	0.06

Single layer sgd	127.92	92.83	0.39	92.47	0.07
Single layer lbfgs	304.50	93.95	0.36	92.40	0.06
20 hidden layer adam	73.22	94.22	0.37	93.26	0.07
20 hidden layer sgd	275.80	97.04	0.43	95.89	0.06
20 hidden layer lbfgs	320.25	94.28	0.42	93.08	0.07
Adam 20_20 hidden layer	45.12	96.45	0.39	95.43	0.07
Adam 20_20 hidden layer log function	78.96	97.92	0.45	95.63	0.07
Adam 20_20 hidden layer identity function	24.94	92.90	0.36	92.36	0.07
Adam 20_20 hidden layer tanh function	33.62	96.56	0.46	94.85	0.07
Adam 50_50 hidden layer tanh function	56.05	99.20	0.62	96.75	0.10
Adam 50_20 hidden layer tanh function	60.08	98.96	0.61	96.76	0.10
Adam 20_50 hidden layer tanh function	39.09	96.09	0.52	94.98	0.10
Adam 50_50_50_50_50 hidden layer tanh function	86.16	99.01	1.04	96.76	0.17

Adam 50_50_50 hidden layer tanh function	**60.28**	**99.02**	**0.77**	**96.79**	**0.14**
Adam 50_20_50 hidden layer tanh function	58.51	98.31	0.72	96.08	0.12
Adam 20_50_20 hidden layer tanh function	42.76	96.99	0.58	95.26	0.10

Comparing the two classifier models:

Overall, we found a slightly higher accuracy with the convolutional neural nets, but the tradeoff of fit time was very significant. The perceptrons took about one minute to fit, while the best CNNs took almost two hours. Some of the CNNs with fewer kernels did still achieve higher accuracies than the MLPs, and they took about 4 minutes to fit. As for evaluation time, the CNNs again took significantly longer. Where the MLPs took less than one second for evaluation of the training data, the CNNs took about 10 seconds for most, but almost a minute for the best. Also, for the test set evaluation, the MLPs timing was in the tenths of seconds range, but the CNNs took between 1 and 30 seconds.

Hypothesis Evaluation

	Hypotheses	Evaluation
CNN	Increasing number of kernels per layer (K) at least up to 32 in the Convolutional layer will increase accuracy rate and fitting time	Our results show the best overall score on the testing set = 98.37% is when K = 32. With more kernels per layer, the more features of each image are considered (from low frequency to high frequency features)
	Increasing the size of the kernel in the Convolutional layer will decrease accuracy rate and increase fitting	When we increased F=5 and F=11, since the kernel size gets bigger, more computation is done during the convolution

	time.	process, thus the fitting time increase. Scores on both the training and testing sets also decrease as kernel size increase because we are generalizing the data, reduce feature locality by summing up over a larger region of each image.
	Increasing number of output nodes for the Dense layer may increase accuracy rate slightly	Accuracy rate was highest at D= 128, for D=256, it went down by .3% and a little lower when D=384
	Increasing the batchsize will increase the accuracy rate and also decrease fitting time	As we vary B=16, 32 and 64 we found that the fitting time increases with B, while accuracy rates on both the training and testing sets were the best when B=32. This could be because when B=32, it allows a good enough number of input nodes to be considered at a time, without being too susceptible to outliers (when B is lower) or over generalized (with higher B).
	Adding a BatchNormalization layer after the Convolutional layer will decrease the discrepancy between the training and testing accuracy rate and may result in better accuracy rate for the testing set	Adding a BatchNormalization increased the fitting time significantly by 1358s, while training and testing accuracy rates were slightly worse, the difference between these 2 rates also slightly decrease.
MLP	Increasing number of max iterations will allow the classifier to reach equilibrium and thus increase accuracy rate	We observed slightly better performance with larger number of iterations. For example, with 2 hidden layers each of 20 output nodes, tanh activation function, accuracy rate reached 94.85% with 500 max iterations, but only reached 94.07% with 10 max iterations.
	Increasing number of output nodes in each layer and number of layers will increase accuracy rate for the training set, but may be prone to overfitting and thus lower score on the testing set	This varies, we observed better accuracy rate on the testing set when we increase number of output nodes from 2 hidden layers (20,20) to (20,50), however it slightly decrease when we changed from (20, 20) to (50,20). However, when we increased the number of layers to 3, from (50, 50), to (50, 50, 50), accuracy rate slightly increases and got the best overall all MLP test, 96.79%.
	Since “relu” and “adam” are the	For the choice of optimizer function,

	default options for activation and optimizer functions respectively we expected that they will give the best performance on the training and the testing sets	“adam” wasn’t the one performing best with single layer (20), instead it was lbfgs gave the highest score of 95.89% on the testing set. Testing on the MLP(20, 20), with different activation functions showed that ‘relu’ wasn’t the one giving the best accuracy rate on the testing set, “tanh” gave the best score comparing to other activation functions, 94.85%.
CNN vs. MLP	The runtime for CNN will be higher, but the accuracy will be better	Comparing the 2 models with its best performing hyperparameters, with much more number of layers, and output nodes in the fully connected layer, CNN’s fitting process took much longer than that of MLP (5249.60s vs. 60.28). Accuracy rate on the testing set, however, was higher for CNN 98.37%, comparing with MLP’s 96.79%. This could be because of the increase in number of hidden layers, and a better combination of convolutional layers (which gives more local feature evaluation) and fully-connected layer in the CNN, comparing to just fully-connected layers in MLP.

Importance of Testing

It is important to validate the classifier on a separate dataset from the one it was trained on in order to avoid overfitting. If you trained a classifier that had been overfit, it would work better on the training data than the test data. If it were grossly overfit, it would attain 100% on the training data, and would not be able to classify data it hadn’t been trained on because the classifier would be trained exactly to the training data.
In some cases, we did see an almost 3% difference in accuracy for the MLPs, but we checked that this was not overfitting by checking with even more layers and observing similar characteristics. For example, the three-layer 50-node MLP and the five-layer 50-node MLP had very similar accuracy ratios, indicating that the three-layer one was not likely overfitting the data despite having 50^3 = 125000 weights.

Max iterations for MLP

Additional data for MLP, with varying numbers of iterations. The parameter max_iter allows us to select how many iterations the classifier may go through in order to finalize the weights. This allowed us to identify which hyperparameters were very slow to train, so that we did not waste our time improving them. The following three tables summarize the perceptrons we tested with three different maximum iteration values: 10, 50, and 100. In each of these, at least one of the functions in the table did not reach equilibrium for the weights. The data in our analysis above relies on a max_iter value of 500, for which all the classifiers reached a point at which the training loss did not improve beyond the tolerance of 0.0001 for two consecutive iterations (epochs), meaning that the weights reached a sort of equilibrium point.

Max_iter = 10

Parameters	Fit Time	Training Set Accuracy	Training Set Eval Time	Test Set Accuracy	Test Set Eval Time
Single layer adam	12.28	92.20	0.49	92.05	0.09
Single layer sgd	12.30	90.36	0.51	90.89	0.08
Single layer lbfgs	9.67	81.33	0.51	82.46	0.08
20 hidden layer adam	14.79	92.36	0.54	92.15	0.10
20 hidden layer sgd	13.89	90.15	0.54	90.47	0.09
20 hidden layer lbfgs	10.89	52.20	0.55	52.67	0.09
Adam 20_20 hidden layer	16.08	93.61	0.55	92.91	0.10
Adam 20_20 hidden layer log function	13.89	93.70	0.40	93.40	0.07
Adam 20_20 hidden layer identity function	10.25	91.68	0.38	91.00	0.06

Adam 20_20 hidden layer tanh function	10.67	95.14	0.46	94.07	0.07
Adam 50_50 hidden layer tanh function	15.30	97.70	0.64	96.37	0.10
Adam 50_20 hidden layer tanh function	14.45	97.58	0.56	96.42	0.09
Adam 20_50 hidden layer tanh function	11.49	95.77	0.49	95.02	0.10

Max_iter = 50

Parameters	Fit Time	Training Set Accuracy	Training Set Eval Time	Test Set Accuracy	Test Set Eval Time
Single layer adam	29.38	93.10	0.44	92.50	0.08
Single layer sgd	42.23	92.08	0.33	92.18	0.07
Single layer lbfgs	30.26	91.50	0.35	91.60	0.05
20 hidden layer adam	48.77	93.80	0.40	93.15	0.05
20 hidden layer sgd	49.97	93.30	0.38	93.19	0.07
20 hidden layer lbfgs	33.29	86.60	0.36	87.16	0.06
Adam 20_20 hidden layer	45.88	96.45	0.37	95.43	0.07
Adam 20_20 hidden layer log function	52.88	97.43	0.43	95.34	0.08

Adam 20_20 hidden layer identity function	25.15	92.90	0.37	92.36	0.06
Adam 20_20 hidden layer tanh function	34.17	96.56	0.44	94.85	0.08
Adam 50_50 hidden layer tanh function	56.23	99.20	0.64	96.75	0.11
Adam 50_20 hidden layer tanh function	57.50	98.96	0.59	96.76	0.10
Adam 20_50 hidden layer tanh function	37.47	96.09	0.52	94.98	0.08

Max_iter = 100

Parameters	Fit Time	Training Set Accuracy	Training Set Eval Time	Test Set Accuracy	Test Set Eval Time
Single layer adam	28.82	93.10	0.36	92.50	0.07
Single layer sgd	83.70	92.54	0.40	92.09	0.06
Single layer lbfgs	62.45	92.81	0.37	92.19	0.07
20 hidden layer adam	75.69	94.22	0.39	93.26	0.07
20 hidden layer sgd	95.58	94.79	0.37	94.32	0.06
20 hidden layer lbfgs	64.12	90.81	0.44	90.62	0.07
Adam 20_20 hidden layer	44.95	96.45	0.39	95.43	0.07
Adam 20_20	78.88	97.92	0.41	95.63	0.08

hidden layer log function					
Adam 20_20 hidden layer identity function	25.25	92.90	0.35	92.36	0.07
Adam 20_20 hidden layer tanh function	35.64	96.56	0.44	94.85	0.07
Adam 50_50 hidden layer tanh function	58.38	99.20	0.64	96.75	0.10
Adam 50_20 hidden layer tanh function	60.88	98.96	0.62	96.76	0.10
Adam 20_50 hidden layer tanh function	38.30	96.09	0.53	94.98	0.09